Symbolic reductions of probabilistic models using linear process equations

Mark Timmer
February 25, 2010

Joint work with Joost-Pieter Katoen, Jaco van de Pol, and Mariëlle Stoelinga
Contents

1 Introduction

2 A process algebra with data and probability: prCRL

3 Linear probabilistic process equations

4 Case study: leader election protocol

5 Confluence reduction

6 Conclusions and Future Work
Probabilistic Model Checking

Probabilistic model checking:

- Verifying quantitative properties,
- Using a probabilistic model (e.g., a probabilistic automaton)
Probabilistic Model Checking

Probabilistic model checking:
- Verifying quantitative properties,
- Using a probabilistic model (e.g., a probabilistic automaton)

![Diagram of a probabilistic automaton with states and transitions]

- Non-deterministically choose one of the three transitions
- Probabilistically choose the next state

Applications:
- Dependability analysis
- Performance analysis
Limitations and Solutions

Limitations of previous approaches:

- Susceptible to the state space explosion problem
- Restricted treatment of data
Limitations and Solutions

Limitations of previous approaches:
- Susceptible to the state space explosion problem
- Restricted treatment of data

Our approach:

Probabilistic specification (prCRL)

Linear Probabilistic Process Equation (LPPE)

State space (PA)

- Linearisation
- Instantiation
- Optimisation
  - Dead variables
  - Confluence

Visualisation

Model checking
Overview of our approach

Our approach:

1. Specify systems in prCRL: a probabilistic process algebra incorporating both data types and probabilistic choice.
2. Transform specifications to LPPEs: a linear format enabling symbolic optimisations at the language level.
3. Reduce state spaces before they are generated by manipulations of the linear format.
Overview of our approach

Our approach:

1. Specify systems in prCRL: a probabilistic process algebra incorporating both data types and probabilistic choice.
2. Transform specifications to LPPEs: a linear format enabling symbolic optimisations at the language level.
3. Reduce state spaces before they are generated by manipulations of the linear format: confluence reduction.
Contents

1 Introduction

2 A process algebra with data and probability: prCRL

3 Linear probabilistic process equations

4 Case study: leader election protocol

5 Confluence reduction

6 Conclusions and Future Work
A process algebra with data and probability: prCRL

Specification language prCRL:
- Based on \( \mu \text{CRL} \) (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable
A process algebra with data and probability: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable

The grammar of prCRL process terms

Process terms in prCRL are obtained by the following grammar:

\[ p ::= Y(t) \mid c \Rightarrow p \mid p + p \mid \sum_{x:D} p \mid a(t)\sum_{x:D} f : p \]

Process equations and processes

A process equation is something of the form \( X(\vec{g} : \vec{G}) = p \).
An example specification

Sending an arbitrary natural number

\[ X(\text{active} : \text{Bool}) = \]

\[ \text{not}(\text{active}) \Rightarrow \text{ping} \cdot \sum_{b : \text{Bool}} X(b) \]

\[ + \text{active} \Rightarrow \tau \sum_{n : \mathbb{N} > 0} \frac{1}{2^n} : \left( \text{send}(n) \cdot X(\text{false}) \right) \]
An example specification

Sending an arbitrary natural number

\[ X\text{(active : Bool)} = \]
\[ \text{not(active)} \Rightarrow \text{ping} \cdot \sum_{b:\text{Bool}} X(b) \]
\[ + \text{active} \Rightarrow \tau \sum_{n:\mathbb{N}>0} \frac{1}{2^n} : \left( \text{send}(n) \cdot X(\text{false}) \right) \]
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m^2) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m^2) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = (X(1) \parallel Y(2))
\]
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m^2) \cdot Y(m) + \text{choose'} \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = (X(1) \parallel Y(2))
\]

\[\gamma(\text{choose, choose'}) = \text{chooseTogether}\]
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m^2) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = \partial_{\text{choose}, \text{choose}'} (X(1) \parallel Y(2))
\]

\[
\gamma(\text{choose}, \text{choose}') = \text{chooseTogether}
\]
Compositionality using extended prCRL

For compositionality we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} \cdot X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m^2) \cdot Y(m) + \text{choose'} \sum_{m' : \{1, 2\}} \frac{1}{2} \cdot Y(m')
\]

\[
Z = \partial_{\{\text{choose, choose'}\}}(X(1) \parallel Y(2))
\]

\[
\gamma(\text{choose, choose'}) = \text{chooseTogether}
\]
Contents

1. Introduction

2. A process algebra with data and probability: prCRL

3. Linear probabilistic process equations

4. Case study: leader election protocol

5. Confluence reduction

6. Conclusions and Future Work
A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

\[
X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow a_1(\vec{b}_1) \sum_{\vec{e}_1 : \vec{E}_1} f_1 : X(\vec{n}_1) \\
\ldots \\
+ \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(\vec{b}_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k)
\]

- \(\vec{G}\) is a type for state vectors
- \(\vec{D}_i\) a type for local variable vectors for summand \(i\)
- \(c_i\) is the enabling condition of summand \(i\)
- \(a_i\) is an atomic action, with action-parameter vector \(b_i\)
- \(\vec{n}_i\) is the next-state vector of summand \(i\).
- \(\vec{E}_i\) a type for the probabilistic variable for summand \(i\)
- \(f_i\) is the probability distribution of summand \(i\)
Advantages of using LPPEs instead of prCRL specifications:

- Easy state space generation
- Straight-forward parallel composition
- **Symbolic optimisations** enabled at the language level

---

Theorem

Every specification $S$ (without unguarded recursion) can be linearised to an LPPE $S'$ in such a way that $S$ and $S'$ are strongly probabilistic bisimilar.
Advantages of using LPPEs instead of prCRL specifications:

- Easy state space generation
- Straight-forward parallel composition
- Symbolic optimisations enabled at the language level

**Theorem**

*Every specification \( S \) (without unguarded recursion) can be linearised to an LPPE \( S' \) in such a way that \( S \) and \( S' \) are strongly probabilistic bisimilar.*
Linear probabilistic process equations – An example

Specification in prCRL

\[ X(\text{active} : \text{Bool}) = \]
\[ \text{not(active)} \Rightarrow \text{ping} \cdot \sum_{b:\text{Bool}} X(b) \]
\[ + \text{active} \Rightarrow \tau \sum_{n:\mathbb{N} \geq 0} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false}) \]
Linear probabilistic process equations – An example

Specification in prCRL

\[ X(\text{active} : \text{Bool}) = \]
\[ \text{not( active)} \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b) \]
\[ + \text{active} \Rightarrow \tau \sum_{n: \mathbb{N}^\geq 0} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false}) \]

Specification in LPPE

\[ X(pc : \{1..3\}, n : \mathbb{N}^\geq 0) = \]
\[ + pc = 1 \Rightarrow \text{ping} \cdot X(2, 1) \]
\[ + pc = 2 \Rightarrow \text{ping} \cdot X(2, 1) \]
\[ + pc = 2 \Rightarrow \tau \sum_{n: \mathbb{N}^\geq 0} \frac{1}{2^n} : X(3, n) \]
\[ + pc = 3 \Rightarrow \text{send}(n) \cdot X(1, 1) \]
Contents

1. Introduction
2. A process algebra with data and probability: prCRL
3. Linear probabilistic process equations
4. Case study: leader election protocol
5. Confluence reduction
6. Conclusions and Future Work
Case study: a leader election protocol

- **Implementation** in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification

- Manual **dead variable reduction**
Case study: a leader election protocol

- **Implementation** in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification
- Manual dead variable reduction

---

Case study

Leader election protocol à la Itai-Rodeh

- Two processes throw a **die**
  - *One of them throws a 6* → *this will be the leader*
  - *Both throw 6 or neither throws 6* → *throw again*
Case study: a leader election protocol

- **Implementation** in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification
- Manual **dead variable reduction**

**Case study**

Leader election protocol à la Itai-Rodeh

- Two processes throw a **die**
  - *One of them throws a 6 → this will be the leader*
  - *Both throw 6 or neither throws 6 → throw again*

- More precise:
  - *Passive thread: receive value of opponent*
  - *Active thread: roll, send, compare (or block)*
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : Die, set : Bool) =
\]
A prCRL model of the leader election protocol

\[ P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) = \]
\[ \text{set} = \text{false} \Rightarrow \sum_{d:\text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) = \\
set = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
+ set = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) = \\
\quad set = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
\quad + set = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]

\[
A(id : \{1, 2\}) = 
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : Die, set : Bool) = \\
set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
+ set = true \Rightarrow getVal(val) \cdot P(id, val, false)
\]

\[
A(id : \{1, 2\}) = \\
roll(id) \sum_{d:Die} \frac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot 
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) =
\]
\[
set = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true})
\]
\[
+ set = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]

\[
A(id : \{1, 2\}) =
\]
\[
\text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send(\text{other}(id), id, d)} \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) =
\begin{align*}
& \text{set} = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
& + \text{set} = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\end{align*}
\]

\[
A(id : \{1, 2\}) =
\begin{align*}
& \text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot \\
& (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id)) \\
& + (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id)) \\
& + (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id))
\end{align*}
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) =
\]
\[
\begin{align*}
  \text{set} = \text{false} & \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
  \text{set} = \text{true} & \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\end{align*}
\]

\[
A(id : \{1, 2\}) =
\]
\[
\text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot \\
( (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id)) \\
+ (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id)) \\
+ (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id)))
\]

\[
C(id : \{1, 2\}) = P(id, \text{heads}, \text{false}) \parallel A(id)
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) =
\]
\[
\text{set} = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true})
\]
\[
+ \text{set} = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]

\[
A(id : \{1, 2\}) =
\]
\[
\text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot
\]
\[
( (d = e \vee (d \neq 6 \land e \neq 6) \Rightarrow A(id))
\]
\[
+ (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id))
\]
\[
+ (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id))
\]

\[
C(id : \{1, 2\}) = \quad P(id, \text{heads}, \text{false}) \parallel A(id)
\]

\[
\gamma(\text{getVal}, \text{readVal}) = \text{checkVal}
\]
A prCRL model of the leader election protocol

\[
\begin{align*}
P(id : \{1, 2\}, \text{val} : \text{Die}, \text{set} : \text{Bool}) &= \\
\quad \text{set} = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
\quad + \text{set} = \text{true} \Rightarrow \text{getVal}(\text{val}) \cdot P(id, \text{val}, \text{false}) \\
A(id : \{1, 2\}) &= \\
\quad \text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot \\
\quad (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id)) \\
\quad + (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id)) \\
\quad + (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id)) \\
C(id : \{1, 2\}) &= \partial_{\text{getVal}, \text{readVal}}(P(id, \text{heads}, \text{false}) \parallel A(id)) \\
\gamma(\text{getVal}, \text{readVal}) &= \text{checkVal}
\end{align*}
\]
A prCRL model of the leader election protocol

\[
\begin{align*}
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) &= \\
set = \text{false} &\Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
+ \set = \text{true} &\Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\end{align*}
\]

\[
A(id : \{1, 2\}) = \\
\text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot \\
( (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id)) \\
+ (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id)) \\
+ (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id)))
\]

\[
C(id : \{1, 2\}) = \partial_{\text{getVal}, \text{readVal}}(P(id, \text{heads}, \text{false}) \parallel A(id))
\]

\[
S = C(1) \parallel C(2)
\]

\[
\gamma(\text{getVal}, \text{readVal}) = \text{checkVal}
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) =
\]

\[
\text{set} = \text{false} \Rightarrow \sum_{d : \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true})
\]

\[
+ \text{set} = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]

\[
A(id : \{1, 2\}) =
\]

\[
\text{roll}(id) \sum_{d : \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e : \text{Die}} \text{readVal}(e) \cdot
\]

\[
( (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id))
\]

\[
+ (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id))
\]

\[
+ (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id))
\]

\[
C(id : \{1, 2\}) = \partial_{\text{getVal}, \text{readVal}}(P(id, \text{heads}, \text{false}) \parallel A(id))
\]

\[
S = C(1) \parallel C(2)
\]

\[
\gamma(\text{rec, send}) = \text{comm} \quad \gamma(\text{getVal, readVal}) = \text{checkVal}
\]
A prCRL model of the leader election protocol

\[
P(id : \{1, 2\}, val : \text{Die}, set : \text{Bool}) = \\
\quad set = \text{false} \Rightarrow \sum_{d: \text{Die}} \text{rec}(id, \text{other}(id), d) \cdot P(id, d, \text{true}) \\
\quad + set = \text{true} \Rightarrow \text{getVal}(val) \cdot P(id, val, \text{false})
\]

\[
A(id : \{1, 2\}) = \\
\quad \text{roll}(id) \sum_{d: \text{Die}} \frac{1}{6} : \text{send}(\text{other}(id), id, d) \cdot \sum_{e: \text{Die}} \text{readVal}(e) \cdot \\
\quad \left( (d = e \lor (d \neq 6 \land e \neq 6) \Rightarrow A(id)) \\
\quad + (d = 6 \land e \neq 6 \Rightarrow \text{leader}(id) \cdot A(id)) \\
\quad + (e = 6 \land d \neq 6 \Rightarrow \text{follower}(id) \cdot A(id)) \right)
\]

\[
C(id : \{1, 2\}) = \partial_{\text{getVal}, \text{readVal}}(P(id, heads, false) \parallel A(id))
\]

\[
S = \partial_{\text{send}, \text{rec}}(C(1) \parallel C(2))
\]

\[
\gamma(\text{rec}, \text{send}) = \text{comm} \quad \gamma(\text{getVal}, \text{readVal}) = \text{checkVal}
\]
Reductions on the leader election protocol model

In order to obtain reductions first linearise
Reductions on the leader election protocol model

In order to obtain reductions first linearise:

\[
\sum_{e21:D} pc21 = 3 \land pc11 = 1 \land set11 \land val11 = e21 \Rightarrow
\]

checkVal(val11) \[ \sum_{(k1,k2):\{\ast\}\times\{\ast\}} multiply(1.0, 1.0) : \]

\[ Z(1, id11, val11, false, 1, 4, id21, d21, e21, pc12, id12, val12, set12, d12, pc22, id22, d22, e22) \]
In order to obtain reductions first linearise:

\[ \sum_{e21 \in D} pc21 = 3 \land pc11 = 1 \land set11 \land val11 = e21 \Rightarrow \]

\[ \text{checkVal(val11)} \sum_{(k1,k2):\{\ast\} \times \{\ast\}} multiply(1.0, 1.0): \]

\[ Z(1, id11, val11, false, 1, 4, id21, d21, e21, pc12, id12, val12, set12, d12, pc22, id22, d22, e22) \]

Before reductions:

- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions
Reductions on the leader election protocol model

In order to obtain reductions first linearise:

\[
\begin{align*}
    pc21 &= 3 \land set11 \implies \\
    checkVal(val11) &\sum_{(k1,k2):\{\ast\}\times\{\ast\}} 1.0: \\
    Z(val11, false, 4, d21, val11, &
    val12, set12, pc22, d22, e22)
\end{align*}
\]

Before reductions:
- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions

After reductions:
- 10 parameters
- 12 summands
Reductions on the leader election protocol model

In order to obtain reductions first linearise:

\[ pc21 = 3 \wedge \text{set11} \Rightarrow \]

\[ \text{checkVal}(val11) \sum_{(k1,k2):\{\ast\}\times\{\ast\}} 1.0: \]

\[ Z(1, false, 4, d21, val11, val12, set12, pc22, d22, e22) \]

Before reductions:
- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions

After reductions:
- 10 parameters
- 12 summands
- 1613 states (-53%)
- 2278 transitions (-58%)
1. Introduction

2. A process algebra with data and probability: prCRL

3. Linear probabilistic process equations

4. Case study: leader election protocol

5. Confluence reduction

6. Conclusions and Future Work
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiently reducing specifications while preserving branching probabilistic bisimulation.
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

**Confluence reduction**: efficiently reducing specifications while preserving branching probabilistic bisimulation.

Intuition:
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

**Confluence reduction**: efficiently reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

![Confluence reduction diagram](image)
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

**Confluence reduction**: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

![Diagram showing confluence reduction]

- \( a \rightarrow^\tau \)
- \( a \rightarrow \)
- \( \tau \rightarrow \)
- \( \tau \rightarrow \)
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiently reducing specifications while preserving branching probabilistic bisimulation.

Intuition:
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

**Confluence reduction**: efficiently reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

![Diagram showing confluence reduction](image-url)
Variants of confluence

Weak confluence

Confluence

Strong confluence
Examples

Reduction based on strong confluence
Examples

Reduction based on strong confluence

![Diagram showing reduction based on strong confluence]
Examples

Reduction based on strong confluence

- Giving $\tau_c$ steps priority works because of the absence of $\tau_c$ loops.

The diagram shows a transition graph with nodes and transitions labeled with symbols such as $a$, $\tau$, and $b$. The graph illustrates the reduction process based on strong confluence.
Examples

Reduction based on strong confluence

Giving $\tau_c$ steps priority works because of the absence of $\tau_c$ loops.
Examples

Reduction based on strong confluence

\[ \begin{align*}
\tau_c & \quad \text{steps priority works because of the absence of } \\
\tau_c & \quad \text{loops.}
\end{align*} \]
Examples

Reduction based on strong confluence

- Giving steps priority works because of the absence of loops.
Examples

Reduction based on strong confluence

\[
\begin{array}{c}
a \xrightarrow{\tau_c} c \\
\end{array}
\]

\[
\begin{array}{c}
c \xrightarrow{\tau_c} a \\
\end{array}
\]

\[
\begin{array}{c}
a \xrightarrow{\tau_c} b \\
\end{array}
\]

\[
\begin{array}{c}
c \xrightarrow{\tau_c} c \\
\end{array}
\]
Examples

Reduction based on strong confluence

\[
\begin{align*}
\tau_c &\quad a \\
\tau_c &\quad \bar{a} \\
\tau_c &\quad \bar{\tau_c} \\
\end{align*}
\]

\[
\begin{align*}
c &\quad \tau_c \\
\tau_c &\quad a \\
\tau_c &\quad b \\
\end{align*}
\]

\[
\begin{align*}
\tau_c &\quad c \\
\end{align*}
\]
Examples

Reduction based on strong confluence

- Reduction based on strong confluence
- Giving $\tau_c$ steps priority works because of the absence of $\tau_c$ loops.
Examples

Reduction based on strong confluence

Giving $\tau_c$ steps priority works because of the absence of $\tau_c$ loops.
Examples

Reduction based on weak confluence

\[
\begin{align*}
\tau &\quad \tau \\
\tau &\quad \tau \\
\tau &
\end{align*}
\]

Here we used the equivalence classes of \( A/\tau \) as nodes.

(None of the blue nodes could be chosen as representative, as none can do both an \( a \) and a \( b \) transition.)
Reduction based on weak confluence

\[ \tau_c \xrightarrow{\tau} a \xrightarrow{\tau} b \]

(Here we used the equivalence classes of \( A / \tau_c \) as nodes. None of the blue nodes could be chosen as representative, as none of them can done both an \( a \) and an \( b \) transition.)
Examples

Reduction based on weak confluence

Here we used the equivalence classes of $A/\tau_c \leftrightarrow a$ as nodes. (None of the blue nodes could be chosen as representative, as none can do both an $a$ and a $b$ transition.)
Examples

Reduction based on weak confluence
Examples

Reduction based on weak confluence

\[
\begin{align*}
\tau_c &\rightarrow \tau \\
\tau &\rightarrow \tau_c
\end{align*}
\]

Here we used the equivalence classes of \( A/\tau_c \) as nodes. (None of the blue nodes could be chosen as representative, as none of them can do both an \( a \) and a \( b \) transition.)
Examples

Reduction based on weak confluence

\[
\begin{array}{c}
\tau_c \\
\tau_c \\
\tau_c \\
\tau_c \\
\end{array}
\]

Here we used the equivalence classes of \( A/\tau_c \) as nodes. (None of the blue nodes could be chosen as representative, as none of them can do both an \( a \) and a \( b \) transition.)
Examples

Reduction based on weak confluence

Here we used the equivalence classes of $A/\tau_c \rightarrow \rightarrow$ as nodes. (None of the blue nodes could be chosen as representative, as none of them can do both an $a$ and a $b$ transition.)
Examples

Reduction based on weak confluence

Here we used the equivalence classes of $A/\tau_c \leftrightarrow \tau_c$ as nodes. (None of the blue nodes could be chosen as representative, as none of them can do both an $a$ and an $b$ transition.)
Examples

Reduction based on weak confluence

Here we used the equivalence classes of $A / \cong_{\tau_c}$ as nodes. (None of the blue nodes could be chosen as representative, as none of them can do both an $a$ and a $b$ transition.)
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives

[Diagram showing a probabilistic model with transitions labeled 'a', 'b', and 'τ'.]
Examples

Reduction based on confluence using representatives

[Diagram showing a symbolic reduction process with nodes and edges labeled with symbols like $\tau$, $a$, and $b$.]
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives

\[
\begin{align*}
\text{Symbolic reductions of probabilistic models} & \\
\text{February 25, 2010 24 / 32}
\end{align*}
\]
Examples

Reduction based on confluence using representatives

\[
\begin{align*}
\tau_c & \quad a \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\tau_c & \quad \tau_c \\
\end{align*}
\]
Examples

Reduction based on confluence using representatives
Examples

Reduction based on confluence using representatives
For simplicity we only consider strong confluence from now on.

Non-probabilistic:
For simplicity we only consider strong confluence from now on.

Non-probabilistic:

\[ a \quad \tau_c \quad \bar{a} \quad \bar{\tau_c} \]

Probabilistic:

\[ \frac{1}{2} a \quad \frac{1}{2} \quad \tau_c \]
Confluence for probabilistic automata

For simplicity we only consider strong confluence from now on.

Non-probabilistic:

\[
\begin{align*}
\text{a} & \quad \tau_c \quad \text{bar a} \\
\text{bar a} & \quad \tau_c \quad \text{a}
\end{align*}
\]

Probabilistic:

\[
\begin{align*}
\text{a} & \quad \frac{1}{2} \quad \frac{1}{2} \\
\text{bar a} & \quad \frac{1}{3} \quad \frac{1}{6} \quad \frac{1}{2}
\end{align*}
\]
For simplicity we only consider strong confluence from now on.

For non-probabilistic and probabilistic automata, the diagrams illustrate the transitions and probabilities involved. The images show the transitions $a$, $\bar{a}$, and their associated probabilities in the probabilistic case.
Why $\tau_c$ steps should have a Dirac distribution

As all states are (potentially) different, no reduction can be obtained.
Why $\tau_c$ steps should have a Dirac distribution
Why $\tau_c$ steps should have a Dirac distribution
Why $\tau_c$ steps should have a Dirac distribution
Why $\tau_c$ steps should have a Dirac distribution
Why $\tau_c$ steps should have a Dirac distribution

As all states are (potentially) different, no reduction can be obtained.
Detecting confluence using LPPEs

Given an LPPE, confluence can be detected using theorem proving.

\[
X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \sum_{\vec{e}_1 : \vec{E}_1} f_1 : X(\vec{n}_1)
\]

\[
\ldots
\]

\[
+ \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k)
\]

To check the first \(\tau\)-summand is confluent, we check whether indeed

- \(|E_1| = 1\), or \(f_1 = 1.0\) for one \(e_i \in E_1\).
- the summand is confluent with all other summands.
Detecting confluence using LPPEs

\[ X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1) \]

\[ + \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k) \]
Detecting confluence using LPPEs

\[ X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1) \]

\[ \quad \ldots \]

\[ + \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k) \]

To prove:

\[ c_1(g, d_1) \land c_k(g, d_k) \rightarrow \]
Detecting confluence using LPPEs

\[ X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1) \]

\[ + \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k) \]

To prove:

\[ c_1(g, d_1) \land c_k(g, d_k) \rightarrow \]

\[ c_k(n_1(g, d_1), d_k) \]
Detecting confluence using LPPEs

\[
X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1)
\]

\[+ \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k)\]

To prove:

\[
c_1(g, d_1) \land c_k(g, d_k) \rightarrow \\
\left(\begin{array}{c}
c_k(n_1(g, d_1), d_k) \\
\land c_1(n_k(g, d_k, e_k), d_1)
\end{array}\right)
\]
Detecting confluence using LPPEs

\[ X(\vec{g} : \vec{G}) = \sum_{\vec{d}_1 : \vec{D}_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1) \]

\[ \ldots \]

\[ + \sum_{\vec{d}_k : \vec{D}_k} c_k \Rightarrow a_k(b_k) \sum_{\vec{e}_k : \vec{E}_k} f_k : X(\vec{n}_k) \]

To prove:

\[ c_1(g, d_1) \land c_k(g, d_k) \rightarrow \]

\[ \left( \begin{array}{c} c_k(n_1(g, d_1), d_k) \\ \land c_1(n_k(g, d_k, e_k), d_1) \\ \land a_k(g, d_k) = a_k(n_1(g, d_1), d_k) \end{array} \right) \]
Detecting confluence using LPPEs

\[ X(\vec{g} : \vec{G}) = \sum_{d_1 : D_1} c_1 \Rightarrow \tau \cdot X(\vec{n}_1) \]
\[ \ldots \]
\[ + \sum_{d_k : D_k} c_k \Rightarrow a_k(b_k) \sum_{e_k : E_k} f_k \cdot X(\vec{n}_k) \]

To prove:

\[ c_1(g, d_1) \land c_k(g, d_k) \rightarrow \]
\[ \left( \begin{array}{c}
  c_k(n_1(g, d_1), d_k) \\
  \land c_1(n_k(g, d_k, e_k), d_1) \\
  \land a_k(g, d_k) = a_k(n_1(g, d_1), d_k) \\
  \land n_k(n_1(g, d_1), d_k, e_k) = n_1(n_k(g, d_k, e_k), d_1)
\end{array} \right) \]
Reducing LPPEs based on confluent $\tau$ steps

After $\tau_c$ steps have been identified, two types of reductions are possible:

1. **Symbolic prioritisation: change the LPPE**
   - Let $c$ be a confluent summand
   - Find a non-confluent summand $n$ such that $c$ is always enabled after executing $n$
   - Change the next state of $n$, basically merging $n$ and $c$

As we only do this for non-confluent summands, loops are avoided.
Reducing LPPEs based on confluent $\tau$ steps

After $\tau_c$ steps have been identified, two types of reductions are possible:

1. **Symbolic prioritisation:** change the LPPE
   - Let $c$ be a confluent summand
   - Find a non-confluent summand $n$ such that $c$ is always enabled after executing $n$
   - Change the next state of $n$, basically merging $n$ and $c$

   As we only do this for non-confluent summands, loops are avoided.

2. **On-the-fly state space generation using representatives**
   - Generate the state space from the LPPE
   - For each transition that is visited, go to the representative of the target state
   - When no representative is known yet, compute it (using a variation on Tarjan’s SCC algorithm)
1. Introduction
2. A process algebra with data and probability: prCRL
3. Linear probabilistic process equations
4. Case study: leader election protocol
5. Confluence reduction
6. Conclusions and Future Work
Conclusions and Future Work

Conclusions / Results

- We developed the process algebra prCRL, incorporating both data and probability.
- We defined a normal form for prCRL, the LPPE; starting point for symbolic optimisations and easy state space generation.
- We generalised reduction techniques from LPEs to the probabilistic case; constant elimination, confluence reduction.
Conclusions and Future Work

Conclusions / Results

- We developed the **process algebra prCRL**, incorporating both **data** and **probability**.
- We defined a **normal form** for prCRL, the **LPPE**; starting point for symbolic optimisations and easy state space generation.
- We **generalised** reduction techniques from LPEs to the probabilistic case; constant elimination, **confluence reduction**

Future work

- Finish work on **confluence reduction**: proofs, case study, implementation
- Develop **additional reduction techniques**
- Generalise **proof techniques** such as cones and foci to the probabilistic case
Questions

Questions?